

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF EXPRESS MAILING

I hereby certify that this paper and the documents and/or fees referred to as attached therein are being deposited with the United States Postal Service on December 21, 1999 in an envelope as "Express Mail Post Office to Addressee" service under 37 CFR §1.10, Mailing Label Number **EL2439110361** U.S. addressed to the Assistant Commissioner for Patents, Washington, DC 20231.

Joyce L. Ferreira
598 U.S. PTO

Attorney Docket No.: SUN1P254/P4195

First Named Inventor: Pelegri-Llopert

UTILITY PATENT APPLICATION TRANSMITTAL (37 CFR. § 1.53(b))

12/21/99

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

☐ Duplicate for
fee processing

Sir: This is a request for filing a patent application under 37 CFR. § 1.53(b) in the name of inventors:
Eduardo Pelegri-Llopert and Laurence P. G. Cable

For: **MECHANISM FOR AUTOMATIC SYNCHRONIZATION OF SCRIPTING VARIABLES**

Application Elements:

- ☒ 18 Pages of Specification, Claims and Abstract
☒ 4 Sheets of **informal** Drawings
☐ Pages Combined Declaration and Power of Attorney - executed

Accompanying Application Parts:

- ☐ Assignment and Assignment Recordation Cover Sheet (recording fee of \$40.00 enclosed)
☐ 37 CFR 3.73(b) Statement by Assignee
☐ Information Disclosure Statement with Form PTO-1449
☐ Copies of IDS Citations
☐ Preliminary Amendment
☒ Return Receipt Postcard
☐ Small Entity Statement(s)
☐ Other:

Fee Calculation (37 CFR § 1.16)

	(Col. 1) NO. FILED	(Col. 2) NO. EXTRA	SMALL ENTITY RATE	FEE	OR	LARGE ENTITY RATE	FEE
BASIC FEE			\$380	\$	OR	\$760	\$760
TOTAL CLAIMS	12 - 20 = 0		x 9	= \$	OR	x 18	= \$0
INDEP CLAIMS	4 - 03 = 1		x 39	= \$	OR	x 78	= \$78
[] Multiple Dependent Claim Presented			\$130	\$		\$260	\$
			Total	\$	OR	Total	\$838

* If the difference in Col. 1 is less than zero, enter "0" in Col. 2.

☒ **PLEASE DEFER FILING FEES AT THIS TIME.**

☒ The Commissioner is authorized to charge any fees beyond the amount enclosed which may be required, or to credit any overpayment, to Deposit Account No. 50-0388 (Order No SUN1P254).

General Authorization for Petition for Extension of Time (37 CFR §1.136)

☒ Applicants hereby make and generally authorize any Petitions for Extensions of Time as may be needed for any subsequent filings. The Commissioner is also authorized to charge any extension fees under 37 CFR §1.17 as may be needed to Deposit Account No. 50-0388.

☒ Please send correspondence to the following address:

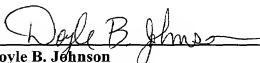
Customer Number: 022434
BEYER & WEAVER, LLP
P.O. Box 61059
Palo Alto, CA 94306



022434

PATENT TRADEMARK OFFICE
U.S. DEPARTMENT OF COMMERCE

Date: December 21, 1999


Doyle B. Johnson
Registration No. 39,240

00471072 122109
651221 2707460

APPLICATION FOR U.S. PATENT

MECHANISM FOR AUTOMATIC SYNCHRONIZATION OF SCRIPTING VARIABLES

INVENTORS: Eduardo Pelegri-Llopart
413 Gilbert Avenue
Menlo Park, Ca 94025

A citizen of Spain

Laurence P. G. Cable

A citizen of the United Kingdom

ASSIGNEE: SUN MICROSYSTEMS, INC.
901 SAN ANTONIO ROAD
PALO ALTO, CA 94303

A DELAWARE CORPORATION

ENTITY: LARGE

BEYER & WEAVER, LLP
P.O. Box 61059
Palo Alto, CA 94306
Telephone (650) 493-2100

MECHANISM FOR AUTOMATIC SYNCHRONIZATION OF SCRIPTING VARIABLES

BACKGROUND OF THE INVENTION

This application claims priority from U.S. Provisional Application Number 60/141,071, filed June 25, 1999, Attorney Docket No. SUN1P249P/P4205PSP, entitled
5 “JAVA SERVER PAGES SPECIFICATION,” and from U.S. Provisional Application Number 60/149,508, filed August 17, 1999, Attorney Docket No. SUN1P260P, entitled “JAVASERVER PAGES SPECIFICATION.” The present application is related to U.S. Patent Application Number _____, entitled “MULTI-LINGUAL TAG
EXTENSION MECHANISM” attorney docket number SUN1P253/P4194, filed
10 _____; the applications are commonly assigned to the assignee of the present invention, and the disclosures of which are herein incorporated by reference.

1. Field of the Invention

The present invention relates generally to the field of computer software, and more particularly to a mechanism for automatic synchronization of scripting variables in a tag
15 extension facility suitable for JavaServer Pages™.

2. Description of the Related Art

JavaServer Pages™ is the Java platform technology for building applications containing dynamic Web content such as HTML, DHTML, XHTML, and XML. A JavaServer Page (JSP) is a text-based document that describes how to process a request to
20 create a response. The description inter-mixes template data with some dynamic actions, taking advantage of the capabilities of the Java platform. The template data is commonly fragments of a structured document (HTML, DHTML, XHTML or XML), and the dynamic actions can be described with scripting elements and/or server-side actiontags.

A simple example of a JSP page is shown in Figure 2. The example shows the response page, which is intended to be a short list of the day of the month and year, at the moment the request is received by the server. The page itself contains some fixed template text, and JSP elements that are shown underlined in the figure. The underlined actions are executed on the server side. When a client makes a request, such as an HTTP request, a request object requests a response from the JavaServer container. The first element creates a Java Bean named clock, of type calendar.jspCalendar. The next two elements use the Bean to display some of its properties (i.e. month and year). The output is sent to a response object which sends a response back to the client.

A JSP page is executed by a JSP container, which is installed on a Web server, or on a Web enabled application server. The JSP container delivers requests from a client to a JSP page and responses from the JSP page to the client. JSP pages may be implemented using a JSP translation or compilation phase that is performed only once, followed by a request processing phase that is performed once per request. The translation phase creates a JSP page implementation class that implements a servlet interface.

Typically, a JSP page contains declarations, fixed template data, action instances that may be nested, and scripting elements. When a request is delivered to a JSP page, all these components are used to create a response object that is then returned to the client. As with standard Web pages, JSP pages may contain "tags." A tag is a textual element within a document that provides instructions for formatting or other actions. For example, World Wide Web documents are set up using HTML (Hyper-Text Mark-up Language) tags which serve various functions such as controlling the styling of text and placement of graphic elements, and also providing links to interactive programs and scripts.

In standard implementations, a JSP page is translated into Java code that runs on a server. The transformation from the JSP page into Java code is done once, when the page is

first accessed. The Java code running on the server is activated when a request is received. In order to create a response object, certain data is passed verbatim to the response, and some actions are executed on the server side. Some of these actions are explicitly spelled out in the JSP page, and other actions are described by the semantics of the text. Thus, there are three types of code: verbatim code, scripting code from the JSP page, and code which has to be defined by the tag library. As used herein, “tags” in the JSP context will be referred to as “actions.”

Ideally, a tag mechanism employed in a JSP page system would allow for actions to be defined in standard libraries. This would allow third parties to provide new actions as part of their JSP authoring tools. For example, as shown in Figure 3, a JSP page author may use a JSP specification-compliant authoring tool to create a Web page. The vendor of the authoring tool can provide new actions via a JSP tag library, such as a tag library that supports chat room functionality. The page can then be deployed into any JSP-compliant container, such as a Web browser. The Web browser uses the same tag library information in order to run the Web page, including the desired chat room functions.

In other words, if a standard JSP tag mechanism is properly defined, vendors of tag libraries can use the standard specification to create tag libraries that are compliant with the JSP environment. Also, vendors of authoring tools can create authoring tools (and scripting languages) compliant with the specification, and vendors of Web browsers can create JSP compliant Web browsers. A Web page author can then choose the best tag library and the best authoring tool available for creating the desired Web pages. Two different Web browsers may support scripting in a completely different manner, but the same tag libraries must be supported by both in order to run the Web page.

In a JSP page, there are objects that are created at execution time, which can be associated with variables in the scripting language. The tag extensions can be invoked from the main JSP page and may update the objects, or create new ones. When creating a new object, the tag extension may need to provide the new object to the JSP page, and associate the object with a scripting variable. However, the tag extensions and the scripting page may be defined in different languages. Thus, there is a need to define a mechanism such that the scripting variables can be passed between the JSP container and the tag extension mechanism and updated automatically (i.e. the variables can be synchronized).

Certain prior extension mechanisms rely on explicit manipulation of the variable to object mapping (i.e. a context). This context is available to both the extension mechanism and to the scripting elements in the JSP page, either directly in the language or through explicit method calls. Other extension mechanisms are mono-lingual and assume that the context is known to both the extension mechanism and the scripting elements in the JSP page. Both approaches are too restrictive, however. Preferably, an implementation would provide greater flexibility for allowing different, compatible scripting languages to work with the tag extension mechanism.

SUMMARY OF THE INVENTION

The present invention defines a mechanism for automatic synchronization of scripting variables in an action tag extension facility. This allows custom actions to create and modify variables, while still allowing the variables to be visible to the scripting language. The values are visible to the scripting code, and the scripting code can modify the values. Also, the values are accessible by the action code. The action code can modify the values or create new values and assign the values to scripting variables so that the scripting code can modify them later.

Attached to each action, there is a TagExtraInfo class that describes the action. The TagExtraInfo method will return at translation time an array of objects that describe the run-time effect of the action. Each object contains a name of each variable as a valid Java object, a Java type of each variable, and a scope parameter which defines a variable's scope with the page and indicates whether the variable is new or pre-existing. At translation-time, the JSP page is translated into a servlet class. In the process of translation, the translator asks for the list of scripting variables from TagExtraInfo object for a given action. If a new variable has been defined, the translator may need to provide a new declaration statement. At run time, the action tag handler references a pageContext object, which provides key values for specified variable names. The pageContext object points to an object associated with a given variable name and is created when the page is executed, thereby associating a variable with a value.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figure 1 is a block diagram of a computer system suitable for implementing the present invention;

Figure 2 illustrates a JSP page;

Figure 3 is a diagram illustrating the relationship between a tag library, a JSP page, and a Web browser; and

Figure 4 is a block diagram illustrating the components of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The following description is provided to enable any person skilled in the art to make and use the invention and sets forth the best modes contemplated by the inventor for carrying out the invention. Various modifications, however, will remain readily apparent to those skilled in the art, since the basic principles of the present invention have been defined herein specifically to provide a mechanism for automatic synchronization of scripting variables in a tag extension facility for JavaServer PagesTM.

The present invention employs various computer-implemented operations involving data stored in computer systems. These operations include, but are not limited to, those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. The operations described herein that form part of the invention are useful machine operations. The manipulations performed are often referred to in terms, such as, producing, identifying, running, determining, comparing, executing, downloading, or detecting. It is sometimes convenient, principally for reasons of common usage, to refer to these electrical or magnetic signals as bits, values, elements, variables, characters, data, or the like. It should be remembered, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

The present invention also relates to a device, system or apparatus for performing the aforementioned operations. The system may be specially constructed for the required purposes, or it may be a general purpose computer selectively activated or configured by a computer program stored in the computer. The processes presented above are not inherently related to any particular computer or other computing apparatus. In particular, various general-purpose computers may be used with programs written in accordance with the

teachings herein, or, alternatively, it may be more convenient to construct a more specialized computer system to perform the required operations.

FIG. 1 is a block diagram of a general purpose computer system 100 suitable for carrying out the processing in accordance with one embodiment of the present invention.

Figure 1 illustrates one embodiment of a general purpose computer system. Other computer system architectures and configurations can be used for carrying out the processing of the present invention. Computer system 100, made up of various subsystems described below, includes at least one microprocessor subsystem (also referred to as a central processing unit, or CPU) 102. That is, CPU 102 can be implemented by a single-chip processor or by multiple processors. It should be noted that in re-configurable computing systems, CPU 102 can be distributed amongst a group of programmable logic devices. In such a system, the programmable logic devices can be reconfigured as needed to control the operation of computer system 100. In this way, the manipulation of input data is distributed amongst the group of programmable logic devices. CPU 102 is a general purpose digital processor which controls the operation of the computer system 100. Using instructions retrieved from memory, the CPU 102 controls the reception and manipulation of input data, and the output and display of data on output devices.

CPU 102 is coupled bi-directionally with a first primary storage 104, typically a random access memory (RAM), and uni-directionally with a second primary storage area 106, typically a read-only memory (ROM), via a memory bus 108. As is well known in the art, primary storage 104 can be used as a general storage area and as scratch-pad memory, and can also be used to store input data and processed data. It can also store programming instructions and data, in the form of data objects, in addition to other data and instructions for processes operating on CPU 102, and is used typically used for fast transfer of data and instructions in a bi-directional manner over the memory bus 108. Also as well known in the

art, primary storage 106 typically includes basic operating instructions, program code, data and objects used by the CPU 102 to perform its functions. Primary storage devices 104 and 106 may include any suitable computer-readable storage media, described below, depending on whether, for example, data access needs to be bi-directional or uni-directional. CPU 102
5 can also directly and very rapidly retrieve and store frequently needed data in a cache memory 110.

A removable mass storage device 112 provides additional data storage capacity for the computer system 100, and is coupled either bi-directionally or uni-directionally to CPU 102 via a peripheral bus 114. For example, a specific removable mass storage device commonly
10 known as a CD-ROM typically passes data uni-directionally to the CPU 102, whereas a floppy disk can pass data bi-directionally to the CPU 102. Storage 112 may also include computer-readable media such as magnetic tape, flash memory, signals embodied on a carrier wave, PC-CARDS, portable mass storage devices, holographic storage devices, and other storage devices. A fixed mass storage 116 also provides additional data storage capacity and
15 is coupled bi-directionally to CPU 102 via peripheral bus 114. The most common example of mass storage 116 is a hard disk drive. Generally, access to these media is slower than access to primary storages 104 and 106.

Mass storage 112 and 116 generally store additional programming instructions, data, and the like that typically are not in active use by the CPU 102. It will be appreciated that the
20 information retained within mass storage 112 and 116 may be incorporated, if needed, in standard fashion as part of primary storage 104 (e.g. RAM) as virtual memory.

In addition to providing CPU 102 access to storage subsystems, the peripheral bus 114 is used to provide access other subsystems and devices as well. In the described embodiment, these include a display monitor 118 and adapter 120, a printer device 122, a network interface

124, an auxiliary input/output device interface 126, a sound card 128 and speakers 130, and other subsystems as needed.

The network interface 124 allows CPU 102 to be coupled to another computer, computer network, or telecommunications network using a network connection as shown.

5 Through the network interface 124, it is contemplated that the CPU 102 might receive information, *e.g.*, data objects or program instructions, from another network, or might output information to another network in the course of performing the above-described method steps. Information, often represented as a sequence of instructions to be executed on a CPU, may be received from and outputted to another network, for example, in the form of a
10 computer data signal embodied in a carrier wave. An interface card or similar device and appropriate software implemented by CPU 102 can be used to connect the computer system 100 to an external network and transfer data according to standard protocols. That is, method embodiments of the present invention may execute solely upon CPU 102, or may be performed across a network such as the Internet, intranet networks, or local area networks, in
15 conjunction with a remote CPU that shares a portion of the processing. Additional mass storage devices (not shown) may also be connected to CPU 102 through network interface 124.

Auxiliary I/O device interface 126 represents general and customized interfaces that allow the CPU 102 to send and, more typically, receive data from other devices such as
20 microphones, touch-sensitive displays, transducer card readers, tape readers, voice or handwriting recognizers, biometrics readers, cameras, portable mass storage devices, and other computers.

Also coupled to the CPU 102 is a keyboard controller 132 via a local bus 134 for receiving input from a keyboard 136 or a pointer device 138, and sending decoded symbols

from the keyboard 136 or pointer device 138 to the CPU 102. The pointer device may be a mouse, stylus, track ball, or tablet, and is useful for interacting with a graphical user interface.

In addition, embodiments of the present invention further relate to computer storage products with a computer readable medium that contain program code for performing various computer-implemented operations. The computer-readable medium is any data storage device that can store data which can thereafter be read by a computer system. The media and program code may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known to those of ordinary skill in the computer software arts. Examples of computer-readable media include, but are not limited to, all the media mentioned above: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and specially configured hardware devices such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs), and ROM and RAM devices. The computer-readable medium can also be distributed as a data signal embodied in a carrier wave over a network of coupled computer systems so that the computer-readable code is stored and executed in a distributed fashion. Examples of program code include both machine code, as produced, for example, by a compiler, or files containing higher level code that may be executed using an interpreter.

It will be appreciated by those skilled in the art that the above described hardware and software elements are of standard design and construction. Other computer systems suitable for use with the invention may include additional or fewer subsystems. In addition, memory bus 108, peripheral bus 114, and local bus 134 are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be used to connect the CPU to fixed mass storage 116 and display adapter 120. The computer system shown in FIG. 1 is but

an example of a computer system suitable for use with the invention. Other computer architectures having different configurations of subsystems may also be utilized.

As the term is used herein, a tag extension mechanism is a specialized sub-language that enables the addition of new or custom actions, thus allowing the JSP “language” to be easily extended in a portable fashion. A typical example would be elements to support embedded database queries. Tag libraries can be used by JSP authors or JSP authoring tools and can be distributed along with JSP pages to any JSP container (i.e. JSP environment and/or engine), such as Web and application servers. The tag extension mechanism of the present invention can be used from JSP pages written using any valid scripting language, although the mechanism itself only assumes a Java RunTime environment.

As shown in Figure 4, a JSP page, many contain scripting variables and actions (tags). These actions or tags can define new scripting variables or update existing variables. The present invention defines a mechanism that allows the actions to create and modify objects, while still allowing the objects to be visible to the scripting language. The values are visible to the scripting code, and the scripting code can modify the values. Also, the values are accessible by the action code. The action code can modify the values or create new values and assign the values to scripting variables so that the scripting code can modify them later.

A tag library constructed according to the present invention is a collection of available action tags. A tag library includes a tag library descriptor (TLD). For each tag, the tag library descriptor includes a tag handler class (i.e. a request time object of the tag), and an optional TagExtraInfo class. Thus, attached to each action, there is a TagExtraInfo class that describes the action. The TagExtraInfo class knows the attributes of the class, including the names of the scripting variables introduced or modified by the action. In other words, the TagExtraInfo class maintains a list of variables defined or changed in an action. More specifically, the TagExtraInfo method will return at translation time an array of objects that describe the run-

time effect of the action. Each object contains a name of each variable as a valid Java object, a Java type of each variable, and a scope parameter which defines a variable's scope with the page and indicates whether the variable is new or pre-existing.

At translation-time, the JSP page is translated into a servlet class. In the process of translation, the translator asks for the list of scripting variables from TagExtraInfo object for a given action. If a new variable has been defined, the translator may need to provide a new declaration statement. At run time, the action tag handler references a pageContext object, which provides key values for specified variable names. The pageContext object points to an object associated with a given variable name and is created when the page is executed, thereby associating a variable with a value.

The scripting details of the JSP page are only known by the translator, and the translator is the only component that knows how to define new scripting variables and how to assign them. The translator generates code that will access the page context object, according to some contract. The contract is described by a combination of specification-defined conventions plus the information provided by the TagExtraInfo object. At run time, the code generated by the translator will look for the name of a variable (i.e. "foo") and assign its value to the scripting variable.

As described above, the JSP container (translator) knows the details of the scripting language of the page, whereas the author of the action tag is unaware of which scripting language will be used. So in order to provide for maximum flexibility and portability, the present invention provides a mechanism that insulates the tag implementation (tag mechanism) from the scripting language details. If both the tag library and the scripting page were written in the same language, such as Java, the variables could be synchronized more easily. However, in order to provide a more general solution, the present invention exposes

the variables in an action tag at translation time via the TagExtraInfo class, and uses a pageContext object to map the variables to values at run time.

Thus, by having a description of the names and types of the scripting variables provided by the tag library, the JSP container is not limited to any particular scripting language or specific scripting language implementation. The key idea is to combine an explicit run-time representation of the context with the translation-time information of the scripting variables affected. The tag extension uses the run-time context to modify and/or create objects. The main JSP page uses the translation-time information to automatically synchronize the scripting variables whenever the main JSP page code is accessed. In other words, the TagExtraInfo object knows which variables are going to be modified, and the scripting language knows how to do the modification. By providing a mechanism for the tag library and the JSP container to share this information, the present invention facilitates the use of many different scripting languages for JSP pages and provides the addition of many new action tags from various vendors.

In one embodiment, the JSP page implementation instantiates (or reuses) a tag handler object for each action in the JSP page. This handler object is a Java object that implements the javax.servlet.jsp.tagext.Tag interface. The tag handler then passes in the values for the variables to a pageContext object. The pageContext object encapsulates implementation dependent features and provides convenience methods, such as getter methods to obtain references to various request-time objects. The pageContext object also provides access to all the implicit objects (including the request object, for instance) to the handler.

When a custom tag introduces some new identifiers, a javax.servlet.jsp.tagext.TagExtraInfo object is involved at JSP translation time (not at request-time). This object indicates the names and types of the scripting variables that will be assigned objects (at request time) by the action. The only responsibility of the tag author is to

indicate this information in the TagExtraInfo object. The corresponding Tag object must add the objects to the pageContext object. It is the responsibility of the JSP translator to automatically supply all the required code to do the "synchronization" between the pageObject values and the scripting variables.

5 An action tag can define one or more objects, and an id attribute may be used to describe the "main" object (if any) defined by the tag. The scope attribute can be used to describe where to introduce the object. The pageContext interface is used to access and modify the scope data.

In a tag of the form:

```
10 <foo attr="one" attr2="two">
    body
</foo>
```

The action foo may define some scripting variables that are available within a body; and it may also define some scripting variables that are defined after the end of the action tag. The
15 defined variables are inserted into the pageContext object and are available through the pageContext object to the code that implements other tags.

Those skilled in the art will appreciate that various adaptations and modifications of the just-described preferred embodiments can be configured without departing from the scope and spirit of the invention. Therefore, it is to be understood that, within the scope of the appended
20 claims, the invention may be practiced other than as specifically described herein.

CLAIMS

What is claimed is:

1 1. A mechanism for automatic synchronization of scripting variables between a page
2 containing action tags, and a tag library, the mechanism comprising:

3 a pageContext object for each page, the pageContext object containing a mapping of
4 scripting variables to values; and

5 a TagExtraInfo object for each action tag, the TagExtraInfo object containing a
6 method that returns a list of available scripting variables and a variable type associated with
7 each variable.

1 2. The mechanism of Claim 1, wherein when the page is translated, a translator
2 consults the TagExtraInfo object to obtain the list of available scripting variables.

1 3. The mechanism of Claim 2, wherein the pageContext object is created when the
2 page is executed (run-time).

1 4. The mechanism of Claim 3, wherein the TagExtraInfo object comprises:

2 a valid Java object name for each variable;

3 a Java type for each variable; and

4 a scope parameter that specifies a variable's scope relative to the page.

1 5. The mechanism of Claim 4, wherein the tag library does not know which scripting
2 language is used to create the page.

1 6. The mechanism of Claim 5, wherein the page is a JavaServer Page.

1 7. A method for automatically synchronizing scripting variables between a page and a
2 tag library, the method comprising:

3 creating for each action tag a TagExtraInfo object that contains a list of available
4 scripting variables and a variable type associated with each variable;

5 translating the page by referring to the list of scripting variables in the TagExtraInfo
6 object associated with each action tag in the page;
7 executing the page; and
8 creating for each page at execution, a pageContext object that contains a mapping of
9 scripting variables to values.

1 8. The method of Claim 7, wherein the TagExtraInfo object comprises:

2 a valid Java object name for each variable;

3 a Java type for each variable; and

4 a scope parameter that specifies a variable's scope relative to the page.

1 9. The method of Claim 8, wherein the tag library does not know which scripting
2 language is used to create the page.

1 10. The method of Claim 9, wherein the page is a JavaServer Page.

1 11. A computer readable medium including computer program code for providing
2 automatic synchronization of scripting variables in a tag extension mechanism, the computer
3 readable medium comprising:

4 computer program code for creating for each page a pageContext object that contains
5 a mapping of scripting variables to values; and

6 computer program code for creating for each action tag a TagExtraInfo object that
7 contains a list of available scripting variables and a variable type associated with each
8 variable.

1 12. A computer system for providing automatic synchronization of scripting variables
2 in a tag extension mechanism, the computer system comprising:

3 a processor; and

4 computer program code operating on the processor for creating for each page a
5 pageContext object that contains a mapping of scripting variables to values; and

- 6 computer program code for creating for each action tag a TagExtraInfo object that
- 7 contains a list of available scripting variables and a variable type associated with each
- 8 variable.

(a)
 (b)
 (c)
 (d)
 (e)
 (f)
 (g)
 (h)
 (i)
 (j)
 (k)
 (l)
 (m)
 (n)
 (o)
 (p)
 (q)
 (r)
 (s)
 (t)
 (u)
 (v)
 (w)
 (x)
 (y)
 (z)

ABSTRACT

The present invention defines a mechanism for automatic synchronization of scripting variables in an action tag extension facility. Attached to each action, there is a TagExtraInfo class that describes the action. The TagExtraInfo class knows the attributes of the class, including the names of the scripting variables introduced or modified by the action. At translation time, a tag handler accesses the information from the TagExtraInfo class for each tag in a page. At run time, a pageContext object is created containing a mapping of scripting variables to values. The values are visible to the scripting code, and the scripting code can modify the values. Also, the values are accessible by the action code. The action code can modify the values or create new values and assign the values to scripting variables so that the scripting code can modify them later. Thus, the present invention allows action tags to be created without explicit knowledge of the scripting language used to create a page.

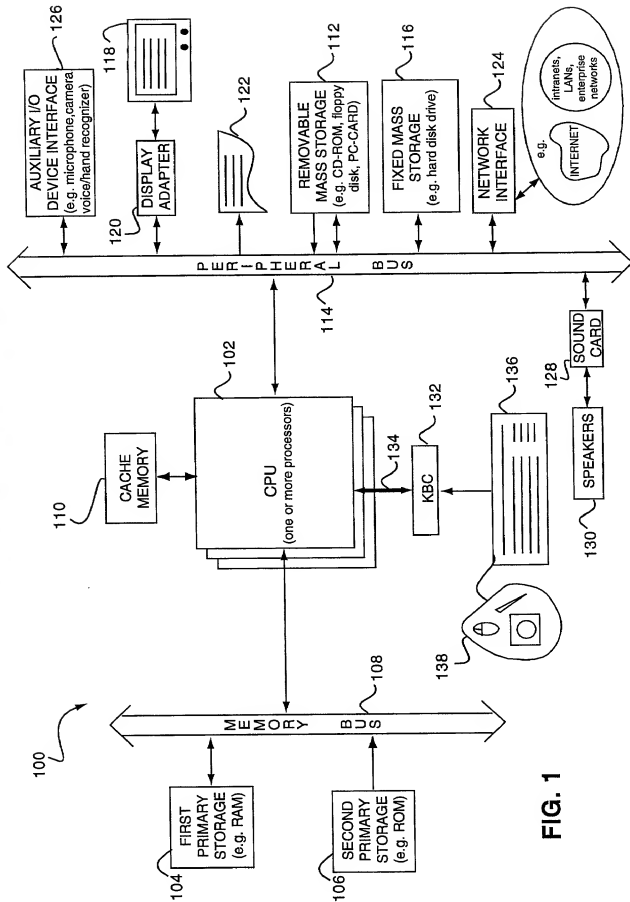


FIG. 1

Client

HTTP

Request

Response

Container

JSP Page

```
<html>
<jsp:useBean id="clock"
              class="calendar.jspCalendar" />
<ul>
<li>Day: <%=clock.getDayOfMonth() %>
<li>Year: <%=clock.getYear() %>
</ul>
</html>
```

Figure 2

JavaServer Pages™
Specification

JSP tag library

Specification-compliant tool

JSP pages
+
HTML pages
+
server side code
+
tag library

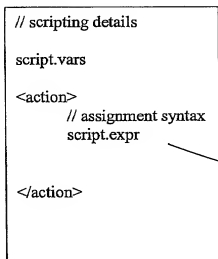
JSP-compliant
container and
Web Browser

JSP tag library

Figure 3

Figure 3 is a diagram illustrating the JSP architecture. It shows the flow from the JavaServer Pages™ Specification and a JSP tag library to a Specification-compliant tool, which then generates JSP pages, HTML pages, server side code, and a tag library. This output is then processed by a JSP-compliant container and Web Browser, which interacts with the JSP tag library.

JSP Page



at run-time

pageContext

foo	val1

translation

TagExtraInfo

1. a valid Java object name for each variable
2. a Java type for each variable
3. a scope parameter that specifies a variable's scope relative to the page.

servlet class

Figure 4